Query Relaxation for Entity-Relationship Search

Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum

Max-Planck Institute for Informatics
{elbass,ramanath,weikum}@mpii.de

Abstract. Entity-relationship-structured data is becoming more important on the Web. For example, large knowledge bases have been automatically constructed by information extraction from Wikipedia and other Web sources. Entities and relationships can be represented by subject-property-object triples in the RDF model, and can then be precisely searched by structured query languages like SPARQL. Because of their Boolean-match semantics, such queries often return too few or even no results. To improve recall, it is thus desirable to support users by *automatically relaxing* or reformulating queries in such a way that the intention of the original user query is preserved while returning a sufficient number of ranked results.

In this paper we describe comprehensive methods to relax SPARQL-like triple-pattern queries in a fully automated manner. Our framework produces a set of relaxations by means of statistical language models for structured RDF data and queries. The query processing algorithms merge the results of different relaxations into a unified result list, with ranking based on any ranking function for structured queries over RDF-data. Our experimental evaluation, with two different datasets about movies and books, shows the effectiveness of the automatically generated relaxations and the improved quality of query results based on assessments collected on the Amazon Mechanical Turk platform.

1 Introduction

1.1 Motivation

There is a trend towards viewing Web or digital-library information in an entity-centric manner: what is the relevant information about a given sports club, a movie star, a politician, a company, a city, a poem, etc. Moreover, when querying the Web, news, or blogs, we like the search results to be organized on a per-entity basis. Prominent examples of this kind of search are entitycube.research.microsoft.com or google.com/squared/. Additionally, services that contribute towards more semantic search are large knowledge repositories, including both handcrafted ones such as freebase.com as well as automatically constructed ones such as trueknowledge.com or dbpedia.org. These have been enabled by knowledge-sharing communities such as Wikipedia and by advances in information extraction (e.g., [2, 6, 17, 23, 20]).

One way of representing entity-centric information, along with structured relationships between entities, is the Semantic-Web data model RDF. An RDF collection consists of a set of subject-property-object (SPO) triples. Each triple is a pair of entities with a named relationship. A small example about books is shown in Table 1.

Table 1. RDF triples

Subject (S)	**Property (P)**	**Object (O)**
Carl_Sagan	wrote	Contact
Carl_Sagan	type	American_Writer
Carl_Sagan	type	Astronomer
Carl_Sagan	bornIn	USA
Carl_Sagan	wonAward	Pulitzer_Prize
Contact	type	novel
Contact	hasGenre	Science_Fiction
Contact	hasTag	aliens
Contact	hasTag	philosopy
Jon_Krakauer	wrote	Into_the_Wild
Into_the_Wild	type	biography
Into_the_Wild	hasTag	adventure
Into_the_Wild	hasTag	wilderness
Jon _Krakauer	hasBestseller	Into_Thin_Air
Jon _Krakauer	citizenOf	USA

RDF data of this kind can be queried using a conjunction of *triple patterns* – the core of SPARQL – where a triple pattern is a triple with variables and the same variable in different patterns denotes a join condition. For example, searching for Pulitzer-prize winning science fiction authors from the USA could be phrased as:

```
?a wrote ?b ; ?b hasGenre Science_Fiction ;
?a wonAward Pulitzer_Prize ; ?a bornIn USA
```

This query contains a conjunction (denoted by ";") of four triple patterns where ?a and ?b denote variables that should match authors and their books respectively.

While the use of triple patterns enables users to formulate their queries in a precise manner, it is possible that the queries are overly constrained and lead to unsatisfactory recall. For example, this query would return very few results even on large book collections, and only one - Carl Sagan - for our example data. However, if the system were able to automatically reformulate one or more conditions in the query, say, replacing bornIn with citizenOf, the system would potentially return a larger number of results.

1.2 Query Relaxation Problem

This paper addresses the *query relaxation* problem: automatically broadening or reformulating triple-pattern queries to retrieve more results without unduly sacrificing precision. We can view this problem as the entity-relationship-oriented counterpart of query expansion in the traditional keyword-search setting. Automatically expanding queries in a robust way so that they would not suffer from topic drifts (e.g., overly broad generalizations) is a difficult problem [3].

The problem of query relaxation for triple-pattern queries has been considered in limited form in [22, 11, 7, 10] and our previous work [8]. Each of these prior approaches focused on very specific aspects of the problem, and only two of them [22, 8] conducted experimental studies on the effectiveness of their proposals. These techniques are discussed in more detail in Sect. 5.

1.3 Our Approach

This paper develops a comprehensive set of query relaxation techniques, where the relaxation candidates can be derived from both the RDF data itself as well as from external ontological and textual sources. Our framework is based on statistical language models (LMs) and provides a principled basis for generating relaxed query candidates. Moreover, we develop a model for holistically ranking results of both the original query and different relaxations into a unified result list.

Our query relaxation framework consists of the following three types of relaxations:

- Entities (subject, object) and relations (property) specified in a triple pattern are relaxed by substituting with related entities and relations. For example, bornIn could be substituted with citizenOf or livesIn and Pulitzer_Prize could be replaced by Hugo_Award or Booker_Prize.
- Entities and relations specified in a triple pattern could be substituted with variables. For example, Pulitzer_Prize could be replaced by ?p to cover arbitrary awards or wonAward could be replaced by ?r, allowing for matches such as nominatedFor and shortlistedFor.
- Triple patterns from the entire query could be either removed or made optional. For example, the triple pattern ?b hasGenre Science_Fiction could be removed entirely, thus increasing the number of authors returned.

The technical contributions of this paper are the following:

- We develop a novel, comprehensive framework for different kinds of query relaxation, in an RDF setting, based on language modeling techniques. Our framework can incorporate external sources such as ontologies and text documents to generate candidate relaxations. Our relaxation framework is described in Sect. 2.
- We develop a general ranking model that combines the results of the original and relaxed queries and utilizes relaxation weights for computing, in a principled manner, query-result rankings. The ranking model is described in Sect. 3.
- We evaluate our model and techniques with two datasets – movie data from imdb.com and book information from the online community librarything.com, and show that our methods provide very good results in terms of NDCG. The results of our user study are reported in Sect. 4.

2 Relaxation Framework

We start by describing the basic setting and some of the terminology used in the rest of this paper.

Knowledge Base. A knowledge base KB of entities and relations is a set of *triples*, where a triple is of the form $\langle e1, r, e2\rangle$ with entities $e1$, $e2$ and relation r (or $\langle s, p, o\rangle$

with subject s, object o, and property p in RDF terminology). An example of such a triple is: Carl_Sagan wrote Contact.

Queries. A query consists of *triple patterns* where a triple pattern is a triple with at least 1 variable. For example, the query "science-fiction books written by Carl Sagan" can be expressed as: Carl_Sagan wrote ?b; ?b hasGenre Science_Fiction consisting of 2 triple patterns.

Given a query with k triple patterns, the result of the query is the set of all k-tuples that are isomorphic to the query when binding the query variables with matching entities and relations in KB. For example, the results for the example query includes the 2-tuple Carl_Sagan wrote Contact;Contact hasGenre Science_Fiction.

2.1 Relaxation Strategy

As mentioned in the introduction, we are interested in three types of relaxations: i) replacing a constant (corresponding to an entity or a relation) in one or more triple patterns of the query with another constant which still reflects the user's intention, ii) replacing a constant in one or more triple patterns with a variable, and iii) removing a triple pattern altogether. In the rest of this section, we describe a framework which incorporates all three relaxations in a holistic manner. Specific details about how the knowledge-base is utilized in the framework are described in Sect. 2.2.

Finding Similar Entities[1]**.** For each entity $\mathsf{E_i}$ in the knowledge base KB, we construct a document $D(\mathsf{E_i})$ (the exact method of doing so will be described in Sect. 2.2). For each document $D(\mathsf{E_i})$, let $LM(\mathsf{E_i})$ be its language model. The similarity between two entities $\mathsf{E_i}$ and $\mathsf{E_j}$ is now computed as the distance between the LMs of the corresponding documents. Specifically, we use the square-root of the Jensen-Shannon divergence (JS-divergence) between two probability distributions (that is, $LM(\mathsf{E_i})$ and $LM(\mathsf{E_j})$, in this case), which is a metric. And so, for an entity of interest E, we can compute a ranked list of similar entities.

Replacing Entities with Variables. We interpret replacing an entity in a triple pattern with a variable as being equivalent to replacing that entity with *any* entity in the knowledge base.

We first construct a special document for the entire knowledge base, $D(\mathsf{KB})$. Let E be the entity of interest (i.e., the entity in the triple pattern to be replaced with a variable). Let $D(\mathsf{E})$ be its document. Now, we construct a document corresponding to "any" entity other than E as: $D(\mathsf{ANY}) = D(\mathsf{KB}) - D(\mathsf{E})$ (i.e., remove the contents of $D(\mathsf{E})$ from $D(\mathsf{KB})$). The similarity between the entity E and "any" entity ANY is computed as the distance between the LMs of their corresponding documents.

In the ranked list of potential replacements for entity E, a variable replacement is now simply another candidate. In other words, the candidates beyond a certain rank are so dissimilar from the given entity, that they may as well be ignored and represented by a single variable.

[1] Note that, even though we refer only to entities in the following, the same applies to relations as well.

Removing Triple Patterns. So far, we gave a high-level description of our relaxation technique for individual entities (or relations) in a triple pattern, without treating the triple pattern holistically. In a given query containing multiple triple patterns, a large number of relaxed queries can be generated by systematically substituting each constant with other constants or variables. We now consider the case when a triple pattern in the query contains only variables. In a post-processing step, we can now choose one of the following options. First, the triple pattern can be made optional. Second, the triple pattern can be removed from the query. To illustrate the two cases, consider the following examples.

Example 1: Consider the query asking for married couples who have acted in the same movie: ?a1 actedIn ?m; ?a2 actedIn ?m; ?a1 marriedTo ?a2 and a relaxation: ?a1 actedIn ?m; ?a2 ?r ?m; ?a1 marriedTo ?a2. Even though the second triple pattern contains only variables, retaining this pattern in the query still gives the user potentially valuable information – that ?a2 was related some how to the movie ?m. Hence, instead of removing this triple pattern from the query, it is only made optional – that is, a result may or may not have a triple which matches this triple pattern.

Example 2: Consider the query asking for movies which James Cameron produced, directed as well as acted in: James_Cameron produced ?m; James_Cameron directed ?m; James_Cameron actedIn ?m and a relaxation: ?x ?r ?m; James_Cameron directed ?m; James_Cameron actedIn ?m. In this case, the first triple pattern matches any random fact about the movie ?m. This does not give any valuable information to the user as in the previous case and can be removed.

2.2 Constructing Documents and LMs

We now describe how to construct documents for entities and relations and how to estimate their corresponding LMs.

Sources of Information. The document for an entity should "describe" the entity. There are at least three different sources of information which we can leverage in order to construct such a document. First, we have the knowledge base itself – this is also the primary source of information in our case since we are processing our queries on the knowledge base. Second, we could make use of external textual sources – for example, we could extract contextual snippets or keywords from text/web documents from which the triples were extracted. Third, we could also utilize external ontologies such as Wordnet, in order to find terms which are semantically close to the entity.

In this paper, our main focus is on utilizing the knowledge base as the information source and hence, we describe our techniques in this context and perform experiments using these techniques. But, our framework can be easily extended to incorporate other information sources and we briefly describe how this can be done at the end of this section.

Documents and LMs for entities. Let E be the entity of interest and $D(\mathsf{E})$ be its document, which is constructed as the set of all triples in which E occurs either as a subject or an object. That is,

$$D(\mathsf{E}) = \{\langle \mathsf{E\ r\ o} \rangle : \langle \mathsf{E\ r\ o} \rangle \in \mathsf{KB}\} \cup \{\langle \mathsf{s\ r\ E} \rangle : \langle \mathsf{s\ r\ E} \rangle \in \mathsf{KB}\}$$

We now need to define the set of terms over which the LM is estimated. We define two kinds of terms: i) "unigrams" U, corresponding to all entities in KB, and, ii) "bigrams" B, corresponding to all entity-relation pairs. That is,

$$U = \{\mathsf{e} : \langle \mathsf{e\ r\ o} \rangle \in \mathsf{KB} || \langle \mathsf{s\ r\ e} \rangle \in \mathsf{KB}\}$$

$$B = \{(\mathsf{er}) : \langle \mathsf{e\ r\ o} \rangle \in \mathsf{KB}\} \cup \{(\mathsf{re}) : \langle \mathsf{s\ r\ e} \rangle \in \mathsf{KB}\}$$

Example: The entity Woody_Allen would have a document consisting of triples Woody_Allen directed Manhattan, Woody_Allen directed Match_Point, Woody_Allen actedIn Scoop, Woody_Allen type Director, Federico_Fellini influences Woody_Allen, etc. The terms in the document would include Scoop, Match_Point, (type,Director), (Federico_Fellini,influences), etc.

Note that the bi-grams usually occur exactly once per entity, but it is still important to capture this information. When we compare the LMs of two entities, we would like identical relationships to be recognized. For example, if for a given entity, we have the bigram (hasWonAward, Academy_Award), we can then distinguish the case where a candidate entity has the term (hasWonAward, Academy_Award) and the term (nominatedFor, Academy_Award). This distinction cannot be made if only unigrams are considered.

Estimating the LM. The LM corresponding to document $D(\mathsf{E})$ is now a mixture model of two LMs: P_U, corresponding to the unigram LM and P_B, the bigram LM. That is,

$$P_{\mathsf{E}}(w) = \mu P_U(w) + (1 - \mu) P_B(w)$$

where μ controls the influence of each component. The unigram and bigram LMs are estimated in the standard way with linear interpolation smoothing from the corpus. That is,

$$P_U(w) = \alpha \frac{c(w; D(\mathsf{E}))}{\Sigma_{w' \in U} c(w'; D(\mathsf{E}))} + (1 - \alpha) \frac{c(w; D(\mathsf{KB}))}{\Sigma_{w' \in U} c(w'; D(\mathsf{KB}))}$$

where $w \in U$, $c(w; D(\mathsf{E}))$ and $c(w; D(\mathsf{KB}))$ are the frequencies of occurrences of w in $D(\mathsf{E})$ and $D(\mathsf{KB})$ respectively and α is the smoothing parameter. The bigram LM is estimated in an analogous manner.

Documents and LMs for relations. Let R be the relation of interest and let $D(\mathsf{R})$ be its document, which is constructed as the set of all triples in which R occurs. That is,

$$D(\mathsf{R}) = \{\langle \mathsf{s'\ R\ o'} \rangle : \langle \mathsf{s'\ R\ o'} \rangle \in \mathsf{KB}\}$$

As with the case of entities, we again define two kinds of terms – "unigrams" and "bigrams". Unigrams correspond to the set of all entities in KB. But, we make a distinction here between entities that occur as subjects and those that occur as objects, since the relation is directional (note that there could be entities that occur as both). That is,

$$S = \{\mathsf{s} : \langle \mathsf{s\ r\ o} \rangle \in \mathsf{KB}\}$$

$$O = \{\mathsf{o} : \langle \mathsf{s\ r\ o} \rangle \in \mathsf{KB}\}$$

$$B = \{(\mathsf{so}) : \langle \mathsf{s\ r\ o} \rangle \in \mathsf{KB}\}$$

Example: Given the relation directed, $D(\text{directed})$ would consist of all triples containing that relation, including, James_Cameron directed Aliens, Woody_Allen directed Manhattan, Woody_Allen directed Match_Point, Sam_Mendes directed American_Beauty, etc. The terms in the document would include James_Cameron, Manhattan, Woody_Allen, (James_Cameron, Aliens), (Sam_Mendes, American_Beauty), etc.

Estimating the LM. The LM of $D(\mathsf{R})$ is a mixture model of three LMs: P_S, corresponding to the unigram LM of terms in S, P_O, corresponding to the unigram LM of terms in O and P_B, corresponding to the bigram LM. That is,

$$P_R(w) = \mu_s P_S(w) + \mu_o P_O(w) + (1 - \mu_s - \mu_o) P_B(w)$$

where μ_s, μ_o control the influence of each component. The unigram and bigram LMs are estimated in the standard way with linear interpolation smoothing from the corpus. That is,

$$P_S(w) = \alpha \frac{c(w; D(\mathsf{R}))}{\Sigma_{w' \in S} c(w'; D(\mathsf{R}))} + (1 - \alpha) \frac{c(w; D(\mathsf{KB}))}{\Sigma_{w' \in S} c(w'; D(\mathsf{KB}))}$$

where $w \in S$, $c(w; D(\mathsf{R}))$ and $c(w; D(\mathsf{KB}))$ are the frequencies of occurrences of w in $D(\mathsf{R})$ and $D(\mathsf{KB})$ respectively, and α is a smoothing parameter. The other unigram LM and the bigram LM are estimated in an analogous manner.

Generating the Candidate List of Relaxations. As previously mentioned, we make use of the square root of the JS-divergence as the similarity score between two entities (or relations). Given probability distributions P and Q, the JS-divergence between them is defined as follows,

$$JS(P||Q) = KL(P||M) + KL(Q||M)$$

where, given two probability distributions R and S, the KL-divergence is defined as,

$$KL(R||S) = \Sigma_j R(j) \log \frac{R(j)}{S(j)}$$

and

$$M = \frac{1}{2}(P + Q)$$

2.3 Examples

Table 2 shows example entities and relations from the IMDB and LibraryThing datasets and their top-5 relaxations derived from these datasets, using the techniques described above. The entry *var* represents the variable candidate. As previously explained, a variable substitution indicates that there were no other *specific* candidates which had a high similarity to the given entity or relation. For example, the commentedOn relation has only one specific candidate relaxation above the variable relaxation – hasFriend. Note that the two relations are relations between people - a person X could comment on something a person Y wrote, or a person X could have a friend Y - whereas the remaining relations are not relations between people. When generating relaxed queries using these individual relaxations, we ignore all candidates which occur after the variable. The process of generating relaxed queries will be explained in Sect. 3.

Table 2. Example entities and relations and their top-5 relaxations

LibraryThing		IMDB	
Egypt	**Non-fiction**	**Academy_Award_for_Best_Actor**	**Thriller**
Ancient_Egypt	Politics	BAFTA_Award_for_Best_Actor	Crime
Mummies	American_History	Golden_Globe_Award_for_Best_Actor_Drama	Horror
Egyptian	Sociology	*var*	Action
Cairo	Essays	Golden_Globe_Award_for_Best_Actor_Musical_or_Comedy	Mystery
Egyptology	History	New_York_Film_Critics_Circle_Award_for_Best_Actor	*var*
wrote	**commentedOn**	**directed**	**bornIn**
hasBook	hasFriend	actedIn	livesIn
hasTagged	*var*	created	originatesFrom
var	hasBook	produced	*var*
hasTag	hasTag	*var*	diedIn
hasLibraryThingPage	hasTagged	type	isCitizenOf

2.4 Using Other Information Sources

The core of our technique lies in constructing the document for an entity E or relation R and estimating its LM. And so, given an information source, it is sufficient to describe: i) how the document is constructed, ii) what the terms are, and, iii) how the LM is estimated. In this paper, we have described these three steps when the information source is the knowledge base of RDF triples. It is easy to extend the same method for other sources. For example, for the case of entities, we could make use of a keyword context or the text documents from which an entity or a triple was extracted. Then a document for an entity will be the set of all keywords or a union of all text snippets associated with it. The terms can be any combination of n-grams and the LM is computed using well-known techniques from the IR literature (see for example, entity LM estimation in [16, 18, 15, 9, 21], in the context of entity ranking).

Once individual LMs have been estimated for an entity or a relation from each information source, a straight-forward method to combine them into a single LM is to use a mixture model of all LMs. The parameters of the mixture model can be set based on the importance of each source. Note that this method does not preclude having different subsets of sources for different entities or relations.

3 Relaxing Queries and Ranking Results

We have so far described techniques to construct candidate lists of relaxations for entities and relations. In this section we describe how we generate relaxed queries and how results for the original and relaxed queries are ranked.

3.1 Generating Relaxed Queries

Let $Q_0 = \{q_1, q_2, ..., q_n\}$ be the query, where q_i is a triple pattern. Let the set of relaxed queries be $R = \{Q_1, Q_2, ..., Q_r\}$, where Q_j is a query with one or more of its triple patterns relaxed. A triple pattern q_i is relaxed by relaxing at least one of its constants. Let s_j be the relaxation score of Q_j, computed by adding up the scores of all entity and relation relaxations in Q_j. Recall that an individual entity or relation score is the square root of the JS-divergence between the LMs of the relaxed entity/relation and the original entity/relation. Clearly, the score s_0 for the original query Q_0 is 0 and the queries can be ordered in ascending order of s_js.

Example: Consider the query asking for Academy award winning action movies and their directors. Table 3 shows the lists L_1, L_2 and L_3 containing the top-3 closest relaxations for each triple pattern along with their scores. Table 4 shows the top-5 relaxed queries and their scores.

Table 3. Top-3 relaxation lists for the triple patterns for an example query

Q: ?x directed ?m; ?m won Academy_Award; ?m hasGenre Action		
L_1	L_2	L_3
?x directed ?m : 0.0	?m won Academy_Award : 0.0	?m hasGenre Action : 0.0
?x actedIn ?m : 0.643	?m won Golden_Globe : 0.624	?m hasGenre Adventure : 0.602
?x created ?m : 0.647	?m won BAFTA_Award : 0.659	?m hasGenre Thriller : 0.612
?x produced ?m : 0.662	?m ?r Academy_Award : 0.778	?m hasGenre Crime : 0.653

Table 4. Top-5 relaxed queries for an example query. The relaxed entities/relations are underlined.

Q: ?x directed ?m; ?m won Academy_Award; ?m hasGenre Action	
Relaxed Queries	**score**
?x directed ?m;?m won Academy_Award;?m hasGenre <u>Adventure</u>	0.602
?x directed ?m;?m won Academy_Award;?m hasGenre <u>Thriller</u>	0.612
?x directed ?m;?m won <u>Golden_Globe</u>;?m hasGenre Action	0.624
?x <u>actedIn</u> ?m;?m won Academy_Award;?m hasGenre Action	0.643
?x <u>created</u> ?m;?m won Academy_Award;?m hasGenre Action	0.647

Now, we describe how results of the original and relaxed queries can be merged and ranked before representing them to the user.

3.2 Result Ranking

Let Q_0 be the original query and let the set of its relaxations be $R = \{Q_1, Q_2, ..., Q_r\}$. Moreover, let the results of the query Q_0 and all its relaxations be the set $\{T_1, T_2, ..., T_n\}$ where T_i is a result matching one (or more) of the queries Q_j. Note that a result T_i can be a match to more than one query. For example, consider the query Q_0 = ?m hasGenre Action and a relaxation Q_j = ?m hasGenre ?x. The result Batman_Begins hasGenre Action matches both Q_0 and Q_j.

To be able to rank the results of the original and relaxed queries, we assume that a result T matching query Q_j is scored using some score function f. The scoring function can be any ranking function for structured triple-pattern queries over RDF-data. In this paper, we make use of the language-model based ranking function described in [8]. Let the score of each result T with respect to query Q_j be $f(T, Q_j)$ and let the score of each relaxed query Q_j be s_j where the score of a relaxed query is computed as described in the previous subsection. In order to merge the results of the original and relaxed queires into a unified result set, we utilize the following scoring function for computing the score of a result T:

$$S(T) = \Sigma_{j=0}^{r} \lambda_j f(T, Q_j)$$

We next describe two techniques to set the values of the different λ's.

Adaptive Weighting. In this weighting scheme, we assume that the user is interested in seeing the results in a holistic manner. That is, a match to a lower ranked (relaxed) query can appear before a match to a higher ranked query. For example, consider the query Q_0 = ?m hasGenre Action and a relaxation Q_j = ?m hasGenre Thriller. The assumption now is that the user would rather see a "famous" movie of genre thriller, rather than an obscure movie of genre action. And so, a "mixing" of results is allowed. To this end, we set the λ_j's as a function of the scores of the relaxed queries s_j's as follows:

$$\lambda_j = \frac{1 - s_j}{\Sigma_{i=0}^{r}(1 - s_i)}$$

Recall that the smaller the s_j is, the closer Q_j is to the original query Q_0. Also recall that s_0 is equal to 0. This weighting scheme basically gives higher weights to the matches to relaxed queries which are closer to the original query. However, matches for a lower ranked query with sufficiently high scores can be ranked above matches for a higher ranked query.

Incremental Weighting. In this weighting scheme, we assume that the user is interested in seeing results *in order*. That is, all ranked matches of the original query first, followed by all ranked matches of the first relaxed query, then those of the second relaxed query, etc. That is, the results are presented "block-wise".

In order to do this, we need to set the λ_j's by examining the scores of the highest scoring and lowest scoring result to a given query. For example, consider our example query : ?m hasGenre Action. Suppose a relaxation to this query is ?x hasGenre Thriller. If we want to ensure that all matches of the original query are displayed before the first match of the relaxed query, we first examine the result with the lowest score for the original query and the highest score for the relaxed query. Let these results be T_0^{low} and T_1^{high}, respectively. We now need to ensure that $\lambda_0 * f(T_0^{low}, Q_0) > \lambda_1 * f(T_1^{high}, Q_1)$.

Note that both incremental as well as adaptive weighting are only two ways in which we can present results to the user. Additional schemes can include a mixture of both schemes for instance, or any other variations. Our ranking model is general enough and can support any number of such fine-grained result presentation schemes.

4 Experimental Evaluation

We evaluated the effectiveness of our relaxation techniques in 2 experiments. The first one evaluated the quality of individual entity and relation relaxations. It also evaluated the quality of the relaxed queries overall. The second experiment evaluated the quality of the final query results obtained from both original and relaxed queries. The complete set of evaluation queries used, relevance assessments collected and an online demo can be found at `http://www.mpii.de/~elbass/demo/demo.html`.

4.1 Setup

All experiments were conducted over two datasets using the Amazon Mechanical Turk service[2]. The first dataset was derived from the LibaryThing community, which is an online catalog and forum about books. The second dataset was derived from a subset of the Internet Movie Database (IMDB). The data from both sources was automatically parsed and converted into RDF triples. Overall, the number of unique entities was over *48,000* for LibraryThing and *59,000* for IMDB. The number of triples was over *700,000* and *600,000* for LibraryThing and IMDB, respectively.

Due to the lack of an RDF query benchmark, we constructed *40* evaluation queries for each dataset and converted them into structured triple-pattern queries. The number of triple patterns in the constructed queries ranged from 1 to 4. Some example queries include: “People who both acted as well as directed an Academy Award winning movie” (3 triple patterns), “Children’s book writers who have won the Booker prize” (3 triple patterns), etc.

4.2 Quality of Relaxations

To evaluate the quality of individual entity and relation relaxations, we extracted all unique entities and relations occurring in all evaluation queries. The total numbers of entities and relations are given in Table 5. For each entity, the top-5 relaxations were retrieved, excluding the variable relaxation. We presented the entity and each relaxation to 6 evaluators and asked them to assess how closely related the two are on a 3-point scale: 2 corresponding to ”closely related”, 1 corresponding to ”related” and 0 corresponding to ”unrelated”. The same was done for each relation.

To evaluate the quality of relaxed queries overall, we generated the top-5 relaxed queries for each evaluation query. The relaxed queries were ranked in ascending order of their scores, which were computed as described in Sect. 3.1. We asked 6 evaluators to assess how close a relaxed query is to the original one on a 4-point scale: 3 corresponding to ”very-close”, 2 to ”close”, 1 to ”not so close” and 0 corresponding to ”unrelated”.

Table 5 shows the results obtained for entity, relation and query relaxations. For a given entity, the average rating for each relaxation was first computed and then this rating was averaged over the top-5 relaxations for that entity. A similar computation was performed for relations and query relaxations. The second row shows the average rating over all relaxed entities, relations and queries. The third row shows the *Pearson*

[2] `http://aws.amazon.com/mturk/`

correlation between the average rating and the JS-divergence. We achieved a strong *negative* correlation for all relaxations which shows that the smaller the score of the relaxation (closer the relaxation is to the original), the higher the rating assigned by the evaluators. The fourth row shows the average rating for the top relaxation.

The fifth and sixth rows in Table 5 show the average rating for relaxations that ranked above and below the variable relaxation respectively. Recall that, for each entity or relation, a possible entry in the relaxation candidate-list is a variable as described in Section 2. For those relaxations that ranked above a variable (i.e., whose JS-divergence is less than that of a variable), the average rating was more than 1.29 for both entities and relations, indicating how close these relaxations are to the original entity or relation. For those relaxations that ranked below a variable, the average rating was less than 1.1 for entities and 0.8 for relations. This shows that the evaluators, in effect, agreed with the ranking of the variable relaxation.

Table 5. Results for entity, relation and query relaxations

Row	**Metric**	**Entities** (0-2)	**Relations** (0-2)	**Queries** (0-3)
1	**No. of items**	87	15	80
2	**Avg. rating**	1.228	0.863	1.89
3	**Correlation**	-0.251	-0.431	-0.119
4	**Avg. rating for top relaxation**	1.323	1.058	1.94
5	**Avg. rating above variable**	*1.295*	*1.292*	-
6	**Avg. rating below variable**	1.007	0.781	-

Table 6. Average NDCG and rating for all evaluation queries for both datasets

Librarything			
	Adaptive	**Incremental**	**Baseline**
NDCG	**0.868**	**0.920**	0.799
Avg. Rating	**2.062**	**2.192**	1.827
IMDB			
	Adaptive	**Incremental**	**Baseline**
NDCG	**0.880**	**0.900**	0.838
Avg. Rating	**1.874**	**1.928**	1.792

4.3 Quality of Query Results

We compared our relaxation framework, with its two weighting scheme variants, *Adaptive* and *Incremental* (see Sect. 3.2), against a baseline approach outlined in [8]. The latter simply replaces a constant in the original query with a variable to generate a relaxed query. The weight of the relaxed triple pattern is determined based on the number of constants replaced. For all 3 methods, we set the weight of an original triple pattern to the same value, to ensure that exact matches would rank on top, and thus make the differences rely solely on the quality and weights of the relaxations. We used the same ranking function f to rank the results with respect to a query Q_j for all 3 techniques. The ranking function was based on the LM-based ranking model in [8].

We pooled the top-10 results from all 3 approaches and presented them to 6 evaluators in no particular order. The evaluators were required to assess the results on a 4 point scale: 3 corresponding to "highly relevant", 2 corresponding to "relevant", 1 corresponding to "somewhat relevant", and 0 corresponding to "irrelevant". To measure the ranking quality of each technique, we used the Normalized Discounted Cumulative Gain (DCG) [12], a standard IR measure that takes into consideration the rank of relevant results and allows the incorporation of different relevance levels.

The results of our user evaluation are shown in Table 6. The reported NDCG values were averaged over all evaluation queries. Both variations of our framework (the first two columns) *significantly* outperformed the baseline approach, with a one-tailed paired t-test (p-value ≤ 0.01). The Adaptive approach had over 8% improvement in NDCG over the baseline for Librarything and over 5% for IMDB. The Incremental approach had improvements over 15% for Librarything and 7% for IMDB. Furthermore, we computed the average rating over all results for each technique as shown in the second and fourth rows (Avg. Rating) in Table 6.

Finally, in Table 7 we show an example evaluation query and the top-3 results returned by each relaxation approach. Next to each result, we show the average rating given by the evaluators. The relaxed constants are underlined.

The example query in Table 7 asks for science fiction books that have tag Film. There is only *one* one such result which is ranked as the top result by all 3 approaches. Since the Adaptive approach ranks the whole set of approximate results, it allows for more diversity in terms of relaxations. And so, the Adaptive approach returns the more famous and iconic movies, Blade and Star_Wars as the top results compared to The_Last_Unicorn and The_Mists_Of_Avalon returned by the Incremental scheme.

Table 7. Top-ranked results for the example query "A science-fiction book that has tag Film"

Result	**Rating**
Q: ?b type Science_Fiction; ?b hasTag Film	
Adaptive	
Star_Trek_Insurrection type Science_Fiction; Star_Trek_Insurrection hasTag Film	2.50
Blade type Science_Fiction; Blade hasTag <u>Movies</u>	2.83
Star_Wars type Science_Fiction; Star_Wars hasTag <u>Made_Into_Movie</u>	2.00
Incremental	
Star_Trek_Insurrection type Science_Fiction; Star_Trek_Insurrection hasTag Film	2.50
The_Last_Unicorn type Science_Fiction; The_Last_Unicorn hasTag <u>Movie/tv</u>	2.50
The_Mists_of_Avalon type Science_Fiction; The_Mists_of_Avalon hasTag <u>Movie/tv</u>	2.17
Baseline	
Star_Trek_Insurrection type Science_Fiction; Star_Trek_Insurrection hasTag Film	2.50
Helter_Skelter type <u>History</u>; Helter_Skelter hasTag Film	0.83
Fear_&_Loathing_in_Vegas type <u>History</u>; Fear_&_Loathing_in_Vegas hasTag Film	1.83

5 Related Work

One of the problems addressed in this paper is that of relaxing entities and relations with similar ones. This is somewhat related to both record linkage [14], and ontology matching [19]. But a key difference is that we are merely trying to find candidates which are *close in spirit* to an entity or relation, and not trying to solve the entity disambiguation problem. Other kinds of reformulations such as spelling correction, etc. directly benefit from techniques for record linkage, but are beyond the scope of our work.

Query reformulation in general has been studied in other contexts such as keyword queries [5] (more generally called query expansion), XML [1, 13], SQL [4, 22] as well as RDF [11, 7, 10]. Our setting of RDF and triple patterns is different in being schema-less (as opposed to relational data) and graph-structured (as opposed to XML which is mainly tree-structured and supports navigational predicates).

For RDF triple-pattern queries, relaxation has been addressed to some extent in [22, 11, 7, 10, 8]. This prior work can be classified based on several criteria as described below. Note that except for [22] and our previous work in [8], none of the other papers report on experimental studies.

Scope of Relaxations. With the exception of [7, 11], the types of relaxations considered in previous papers are limited. For example, [22] considers relaxations of relations only, while [10, 8] consider both entity and relation relaxations. The work in [8], in particular, considers a very limited form of relaxation – replacing entities or relations specified in the triple patterns with variables. Our approach, on the other hand, considers a comprehensive set of relaxations and in contrast to most other previous approaches, weights the relaxed query in terms of the *quality* of the relaxation, rather than the number of relaxations that the query contains.

Relaxation Framework. While each of the proposals mentioned generates multiple relaxed query candidates, the method in which they do so differ. While [7, 11, 10] make use of rule-based rewriting, the work in [22] and our own work make use of the data itself to determine appropriate relaxation candidates. Note that rule-based rewriting requires human input, while our approach is completely automatic.

Result Ranking. Our approach towards result ranking is the only one that takes a holistic view of both the original and relaxed query results. This allows us to rank results based on both the *relevance* of the result itself, as well as the *closeness* of the relaxed query to the original query. The "block-wise" ranking adopted by previous work – that is, results for the original query are listed first, followed by results of the first relaxation and so on – is only one strategy for ranking, among others, that can be supported by our ranking model.

6 Conclusion

We proposed a comprehensive and extensible framework for query relaxation for entity-relationship search. Our framework makes use of language models as its foundation and can incorporate a variety of information sources on entities and relations. We showed how to use an RDF knowledge base to generate high quality relaxations. Furthermore, we showed how different weighting schemes can be used to rank results. Finally, we showed the effectiveness of our techniques through a comprehensive user evaluation. We believe that our contributions are of great importance for an extended-SPARQL API that could underlie the emerging "Web-of-Data" applications such as Linking-Open-Data across heterogeneous RDF sources.

References

[1] Amer-Yahia, S., Lakshmanan, L.V.S., Pandit, S.: Flexpath: Flexible structure and full-text querying for xml. In: SIGMOD (2004)

[2] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A nucleus for a web of open data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)

[3] Billerbeck, B., Zobel, J.: When query expansion fails. In: SIGIR (2003)

[4] Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic information retrieval approach for ranking of database query results. ACM Trans. on Database Syst. 31(3) (2006)

[5] Croft, B., Metzler, D., Strohman, T.: Search Engines: Information Retrieval in Practice. Pearson Education, London (2009)

[6] Doan, A., Gravano, L., Ramakrishnan, R., Vaithyanathan, S. (eds.): Special issue on managing information extraction. ACM SIGMOD Record 37(4) (2008)

[7] Dolog, P., Stuckenschmidt, H., Wache, H., Diederich, J.: Relaxing rdf queries based on user and domain preferences. Journal of Intell. Inf. Sys. (2008)

[8] Elbassuoni, S., Ramanath, M., Schenkel, R., Sydow, M., Weikum, G.: Language-model-based ranking for queries on RDF-graphs. In: CIKM (2009)

[9] Fang, H., Zhai, C.: Probabilistic models for expert finding. In: Amati, G., Carpineto, C., Romano, G. (eds.) ECiR 2007. LNCS, vol. 4425, pp. 418–430. Springer, Heidelberg (2007)

[10] Huang, H., Liu, C., Zhou, X.: Computing relaxed answers on RDF databases. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 163–175. Springer, Heidelberg (2008)

[11] Hurtado, C., Poulovassilis, A., Wood, P.: Query relaxation in rdf. Journal on Data Semantics (2008)

[12] Järvelin, K., Kekäläinen, J.: Ir evaluation methods for retrieving highly relevant documents. In: SIGIR (2000)

[13] Lee, D.: Query Relaxation for XML Model. Ph.D. thesis, UCLA (2002)

[14] Naumann, F., Herschel, M.: An Introduction to Duplicate Detection. Morgan & Claypool, San Francisco (2010)

[15] Nie, Z., Ma, Y., Shi, S., Wen, J.-R., Ma, W.Y.: Web object retrieval. In: WWW (2007)

[16] Petkova, D., Croft, W.: Hierarchical language models for expert finding in enterprise corpora. Int. J. on AI Tools 17(1) (2008)

[17] Sarawagi, S.: Information extraction. Foundations and Trends in Databases 2(1) (2008)

[18] Serdyukov, P., Hiemstra, D.: Modeling documents as mixtures of persons for expert finding. In: Macdonald, C., Ounis, I., Plachouras, V., Ruthven, I., White, R.W. (eds.) ECIR 2008. LNCS, vol. 4956, pp. 309–320. Springer, Heidelberg (2008)

[19] Staab, S., Studer, R.: Handbook on Ontologies (International Handbooks on Information Systems). Springer, Heidelberg (2004)

[20] Suchanek, F., Sozio, M., Weikum, G.: SOFIE: A self-organizing framework for information extraction. In: WWW (2009)

[21] Vallet, D., Zaragoza, H.: Inferring the most important types of a query: a semantic approach. In: SIGIR (2008)

[22] Zhou, X., Gaugaz, J., Balke, W.T., Nejdl, W.: Query relaxation using malleable schemas. In: SIGMOD (2007)

[23] Zhu, J., Nie, Z., Liu, X., Zhang, B., Wen, J.R.: Statsnowball: a statistical approach to extracting entity relationships. In: WWW (2009)